SableCC

26.06.2003

Yasushi Tomii

Überblick

1. **Einleitung**
2. **SableCC**
3. **lexer, Parser**
4. **Framework**
5. **Frage**

1. Einleitung

1.1Was ist SableCC?

ein **Compiler-compiler** für Java.

1.2Was macht SableCC?

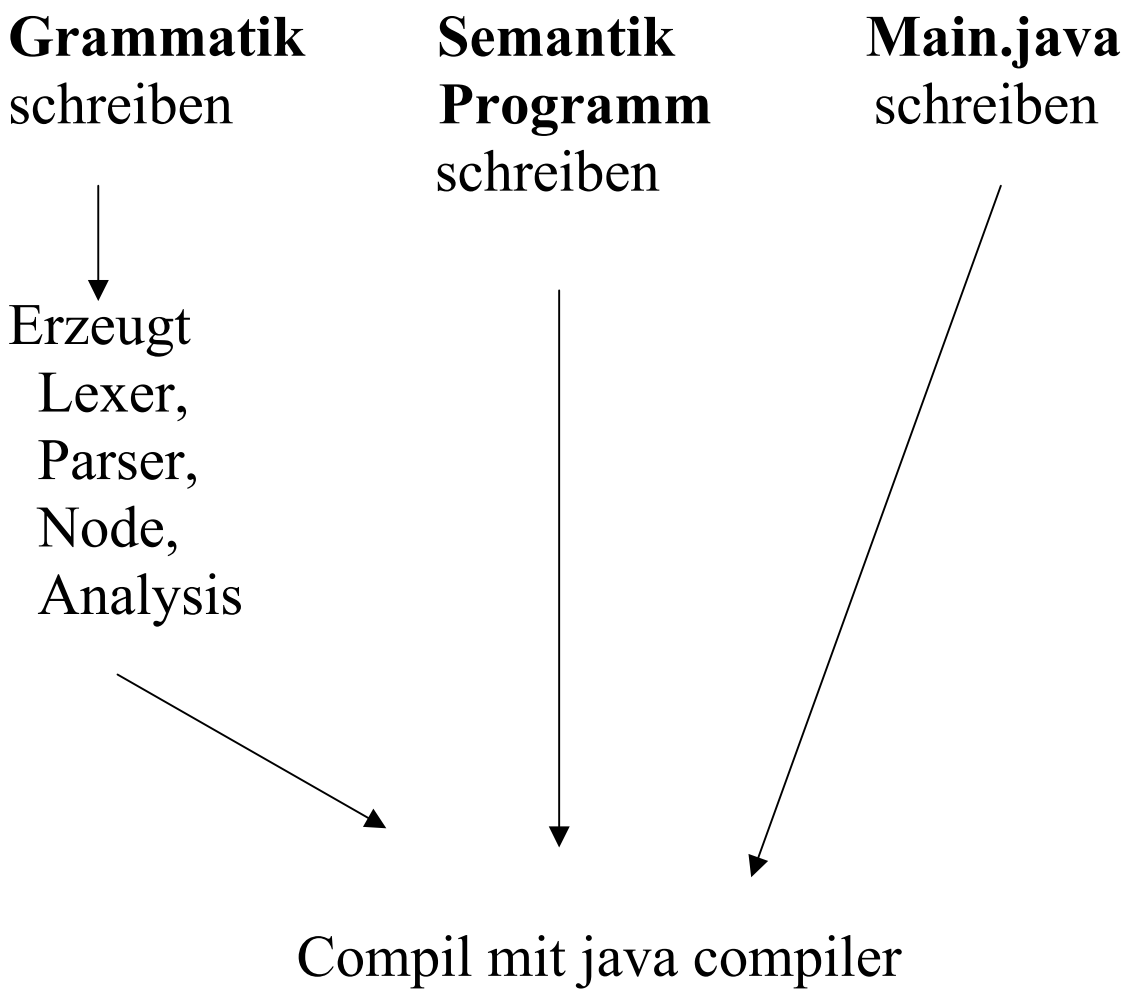
erzeugt

Scanner und **grammatische Definitionen**

für die Javasprache.

(lexer, Parser, Node, Analysis)

1.3Wie Kann man das benutzen?



2. SableCC

2.1 SableCC

Grammatik

Lexer, Parser,
Node, Analysis

2.2 Grammatik

besteht aus

Package, helper**, Token,** State**, Ignored Token**

und **Produktions**

- Package zeigt

Pfad zu Verzeichnis, wo erzeugte Programme sind

- Token

basiert auf **DFA ------> regulär Ausdrück**

besteht aus

1. Element

   Buchstaben a,b,... ,A,B,....
   Dezimal- und Hexadezimal Zahl

2. Menge von Element.

   Verkettung von Element
   Union von Element (wie AUB)
   Rest von Element (wie A \ B)

- Helper, Staate und Ignored Token

sind analog zu Token

- Produktions

basiert auf **LALR(1)** EBNF

***, +, ?** sind erlaubt.

Aber

( ) ist **nicht** erlaubt.

- {xxx} yyy

  node.**xxxYyy**()

- {xxx}[z]:yyy

  get**Z**()Methode in node.xxxYyy()

- **T**xxx

  eine class **für Token** xxx

- **P**xxx

  eine class **für Produktion** xxx

- **A**xxx

  eine class **für einzige produktion** xxx

3. Lexer, Paeser

3.1 Lexer

SableCC erzeugt **Lexer (class)** unter

verzeichnis [pakage].Lexer

lexer arbeitet für **Tokenerkennt** mit Hilfe von

PuchBackReader und **filter()**.

3.2 Paeser

- SableCC erzeugt **Parser (class)** unter

  verzeichnis [pakage].Parser.

- public Parser(root.lexer.Lexer);

  als Konstruktor

- public Start parse() throws ParserException, LexerException, IOException

  für AST

- Def. **Start** = *erste produktion* **EOF**;

4. Framework

- **AST** zu erzeugen

- **node.*** und **analysis.*** classes

- **Visitor Design Pattern**

- **AST Walkers**

Quelle

http//www.sablecc.org

http//iyama.ulis.ac.jp/~shishido/
compiler/library/kudo/tyakusyu.ppt

http//www.cwi.nl/~steven/pascal/

Beispiel-1 Grammatik

```
Package sample;

Helpers
 digit = ['0'..'9'];

Tokens
 number = digit+;
 lparen = '(';
 rparen = ')';
 add = '+';
 sub = '-';
 mul = '*';
 div = '/';
 blank = ' '*;

Ignored Tokens
 blank;

Productions
 add_expression =
   {mul} mul_expression |
   {add} add_expression add mul_expression |
   {sub} add_expression sub mul_expression ;

 mul_expression =
   {unary} unary_expression |
   {mul} mul_expression mul unary_expression |
   {div} mul_expression div unary_expression ;

 unary_expression =
   {number} number |
   {paren} l_paren add_expression r_paren ;
```

Beispiel-3 Translation

```
package sample;
import sample.analysis.*;
import sample.node.*;

class Translation extends DepthFirstAdapter
{
 public void caseTNumber(TNumber node)
 { // When we see a number, we print it.
  System.out.print(node);
 }

 public void outAAddAddExpression
                      (AAddAddExpression node)
 {// out of alternative {add} in Expression, we
  // print the plus.
  System.out.print(node.getPlus());
 }
.......................



}
```

Beispiel-2 Main

```
package sample;
import sample.parser.*;
import sample.lexer.*;
import sample.node.*;
import java.io.*;


public class Main
{
 public static void main(String[] arguments)
 {
  try
  {// Create a Parser instance.
   Parser p =
    new Parser(
    new Lexer(
    new PushbackReader(
    new InputStreamReader(System.in), 1024)));

   // Parse the input.
   Start tree = p.parse();

   // Apply the translation.
   tree.apply(new Translation());
  }
  catch(Exception e)
  {
   System.out.println(e.getMessage());
  }
 }
}
```

Beispiel-4 Erzeugte Klasse

sample/analysis/Analysis.java
sample/analysis/AnalysisAdapter.java
sample/analysis/DepthFirstAdapter.java
sample/analysis/ReversedDepthFirstAdapter.java
sample/lexer/Lexer.java
sample/lexer/LexerException.java
sample/node/AAddAddExpression.java
sample/node/ADivMulExpression.java
sample/node/AMulAddExpression.java
sample/node/AMulMulExpression.java
sample/node/ANumberUnaryExpression.java
sample/node/AParenUnaryExpression.java
sample/node/ASubAddExpression.java
sample/node/AUnaryMulExpression.java
sample/node/Cast.java
sample/node/EOF.java
sample/node/NoCast.java
sample/node/**Node**.java
sample/node/NodeCast.java
sample/node/PAddExpression.java
sample/node/PMulExpression.java
sample/node/PUnaryExpression.java
sample/node/Start.java
sample/node/**Switch**.java
sample/node/**Switchable**.java
sample/node/TAdd.java
sample/node/TDiv.java
sample/node/TLParen.java
sample/node/TMul.java
sample/node/**TNumber**.java
sample/node/TRParen.java
sample/node/TSub.java
sample/node/**Token**.java
sample/node/TypedLinkedList.java
sample/parser/Parser.java
sample/parser/ParserException.java
sample/parser/State.java
sample/parser/TokenIndex.java

Beispiel – 5 Ergebnis

```
C:\sablecc>java -jar lib/sablecc.jar sample1
Verifying identifiers.
Generating token classes.
Generating production classes.
Generating alternative classes.
Generating analysis classes.
Generating utility classes.
Generating the lexer.
-Constructing NFA.
.......................
-Constructing DFA.
............
-resolving ACCEPT states.
Generating the parser.
...................
...................
...................
..
...................
C:\Java>
```

Beispiel - 6

```
package sample.node;

import sample.analysis.*;

public final class TNumber extends Token
{
    public TNumber(String text)
    {
        setText(text);
    }
}

    public TNumber(String text, int line, int pos)
    {
        setText(text);
        setLine(line);
        setPos(pos);

    public Object clone()
    {
      return new Tnumber
                    (getText(), getLine(), getPos());
    }

    public void apply(Switch sw)
    {
        ((Analysis) sw).caseTNumber(this);
    }
}
```

Beispiel -7

```
package sample.analysis;

import java.util.*;
import sample.node.*;

public class DepthFirstAdapter extends
AnalysisAdapter
{
.........................


    public void caseStart(Start node)
    {
        inStart(node);
        node.getPAddExpression().apply(this);
        node.getEOF().apply(this);
        outStart(node);
    }


.....................
}
```